# Worm Detection Using Local Networks

Xinzhou Qin, David Dagon, Guofei Gu, Wenke Lee
Georgia Institute of Technology
{xinzhou, dagon, guofei, wenke}@cc.gatech.edu

Mike Warfield, Pete Allor
Internet Security Systems
{mhw, pallor}@iss.net

## Abstract

The need for a global monitoring system for Internet worm detection is clear. Likewise, the need for local detection and response is also obvious. In this study, we used a large data set to review some of the worm monitoring and detection strategies proposed for large networks, and found them difficult to apply to local networks. In particular, the Kalman filter and victim number-based approaches proved unsuitable for smaller networks. They are of course appropriate for large systems, but what work well for local networks?

We propose two algorithms tailored for local network monitoring needs. First, the Destination Source Correlation (DSC) algorithm focuses on the infection relation, and tracks real infected hosts (and not merely scans) to provide an accurate response. Second, the HoneyStat system provides a way to track the short-term infection behavior used by worms. Potentially, this provides a basis for statistical inference about a worm's behavior on a network.

## 1 Introduction

In recent years, there have been multiple algorithms for worm early detection, e.g., [22, 1, 21]. Much of the recent work has suggested the need for large ($2^{20}$ host) networks to detect worm outbreaks [22, 1, 21]. Thus, many have called for the creation of a cyber "Center for Disease Control" (CDC) [18].

Our study confirms the value of this approach, but there are valid reasons why a CDC model *alone* might require improvement. First, a CDC for one million monitored hosts (or more) will have to overcome significant privacy concerns. Many corporations may be unwilling to share data with each other, and customers privacy concerns will present barriers to those willing to consider such a proposal.

Second, even if there is sufficient buy-in for a CDC model, it may not always provide useful *immediate* information to all participants. While most participants in a collaborative monitoring project would benefit from early warnings, the worm detection game requires victims. Potentially, this means one participant will hear about a worm outbreak only after it has affected his/her network. Similarly a contributor to a CDC monitoring system might have enough local information to conclude there is a worm, based on additional information not shared with others. Timely local warnings can provide a second layer of assurance, and help minimize damage.

All of the approaches for the CDC model require the use of large networks to gather data. For this reason, much of the work so far was based on simulations or small data captures. Using data from a 25,000 node network, we consider whether these CDC-scale approaches can help smaller networks detect worms.

Our analysis first looks at whether the monitoring strategies proposed for the CDC model can be used on smaller networks. Based on a substantial amount of real network data, we conclude the detection strategies used by the larger networks are sometimes problematic, and need refinement. They can work on smaller networks, but were obviously not designed for such scenarios.

We therefore consider other algorithms that local networks can deploy. We have designed two new detection algorithms. The first one focuses on investigation of the worm infection relationship among hosts. Using this algorithm, we can identify the victim in the local networks of interests. Unlike other related work that use illegitimate scans to inactive IP space, our approach analyzes inbound and outbound scans to realistic and active networks. Another approach is to examine the short-term network traffic of infected hosts. With sufficient numbers of honeypots, we logistically determine what type of network traffic contributed to a honeypot's infection.

These techniques work well on a local level, and of course provide another valuable stream of information to a CDC-style monitoring system. The key difference is that with these approaches one does not have to wait for the global monitoring system to count enough victims to act. Local knowledge of directed worm scanning can be acted upon immediately, and shared with others in the monitoring network. Thus, just as the CDC relies on regional networks, we propose the equivalent of local public measures for smaller networks.

The rest of this paper is organized as follows: Section 2 is a discussion of related work. In Section 3, we describe our experiments and performance evaluation of two existing worm detection techniques using *real data*. We describe our Destination-Source Correlation (DSC) algorithm in Section 4. Section 5 presents our statistical-based worm detection technique using honeypots. We summarize our work and point out the future work in Section 6.

## 2 Related Work

In worm propagation study, Kephart et. al [7, 8, 6] proposed an epidemiological model in modeling virus spread in 1991 and 1993. Worm researchers adapted the epidemic propagation model to model the spread of worms. In particular, Staniford [17] first applied a "simple classic epidemic model" to study the Code Red propagation. Zou et. al [23] proposed a modified "two-factor" epidemic model that considered the situation of human countermeasures and congestion due to worm traffic. In [2], Chen et. al proposed a discrete time-based propagation model to track the spread of worms using random scanning methods. In this approach, they also consider the effects of patching and worm cleaning during the worm propagation.

Multiple approaches have been proposed for worm early detection. Staniford et al. [18] first proposed an idea of establishing a cyber "Center for Disease Control" that takes leading roles of identifying worms, counter measuring worm propagation, analyzing new vectors and new worms. In collecting information of worm activities, Moore [12] proposed to set up distributed "network telescopes" using a reasonable large fraction of IP space to observe security events occurring on the Internet. In practice, SANS has established an "Internet Storm Center" that collects security logs from distributed intrusion detection systems (IDS) around the globe [4]. The nature of all the approaches is to take advantages of distributed security sensors to gather security information from a wide cyber territories. Correlation techniques are then applied to analyze information and identify worms.

In area of worm early detection techniques, Zou et. al [22] proposed a Kalman filter-based detection algorithm. This approach detects the trend of illegitimate scans to a large unused IP space. In [21], Wu et. al proposed a victim counter-based detection algorithm that tracks the increased rate of new infected hosts. Worm alerts are output when anomaly events occur consecutively over a certain number of times. Berk [1] proposed to use ICMP "Destination Unreachable" messages collected at border routers to infer worm activities. This approach is based on threshold-based anomaly detection.

Researchers have also used honeypots to distract attackers, early warnings about new attack techniques and in-depth analysis of an adversary's strategies [16, 15]. Traditionally, honeypots have been used to gather intelligence about how human attackers operate [16]. Using honeynets to gather and identify attacks was also proposed and implemented [13]. In [10], researchers used honeypots inside a university to detect infected machines behind a firewall. This augmented an existing IDS and sometimes provided earlier warnings of compromised machines. However, traditional honeypots require the security analyst intensive efforts to review logs that has made them unsuitable for a real-time IDS. Researchers have also considered using virtual honeypots, particularly with honeyd [14] which has been extended to emulate some services, e.g., NetBIOS. Recently, honeyd has been offered as a way to detect and disable worms [14]. We believe this approach has promise, but it has to overcome a few significant hurdles before it is used as an early warning IDS. It is not clear how a daemon emulating a network service can catch *zero day* worms. If one knows a worm's attack pattern, it is possible to write modules that will behave like a vulnerable service. But before this is known, catching zero day worms requires emulating even the *presumably unknown* bugs in a net-

work service. Worms that do anything complex (such as downloading a Trojan) can easily evade honeyd, until a module emulating the bug is created.

Our work using honeypot is different from other related work in that we focus on detecting *zero-day* worms. We apply statistical technique to analyze and correlate port activities and state of honeypots to detect worm activities. We do not count on pattern matching or prior knowledge of worm signatures in the worm detection. Therefore, our work is more analogous to *anomaly detection* in the intrusion detection field.

## 3 Reality Check

Based on the nature of our monitoring networks, we applied two existing worm detection algorithms, i.e., Kalman filter-based early detection model [22] and victim number-based detection algorithm [21], to our *real data* corresponding to multiple worm activities collected from our monitoring systems.

In this section, we describe our experiments and evaluate the performance of these two detection models applied in our networks.

### 3.1 Monitored Networks

In our study, we use a darknet that consists of 100 /24 networks (i.e., 25,600 nodes). The darknet is assigned unused IP addresses and keeps inactive in the Internet. It provides passive logging for inbound traffic and does not have any outbound traffic. Additionally, we also collect data from some 20 /24 (5,120 nodes) live machines, variously deployed as honeypots, honeynets and other interactive systems. In the experiments described in this section, we only use real data collected from our darknet. In particular, we applied the Kalman filter-based algorithms and victim number-based detection model to data corresponding to worms Code Red, SQL Slammer and Blaster respectively.

### 3.2 Kalman Filter-based Approach

Kalman Filter [5] is a set of recursive filtering algorithms that provide an efficient computational solution of the least-squares method. The filter supports estimations of past, present and future states, even when the precise nature of the modeled system is unknown.

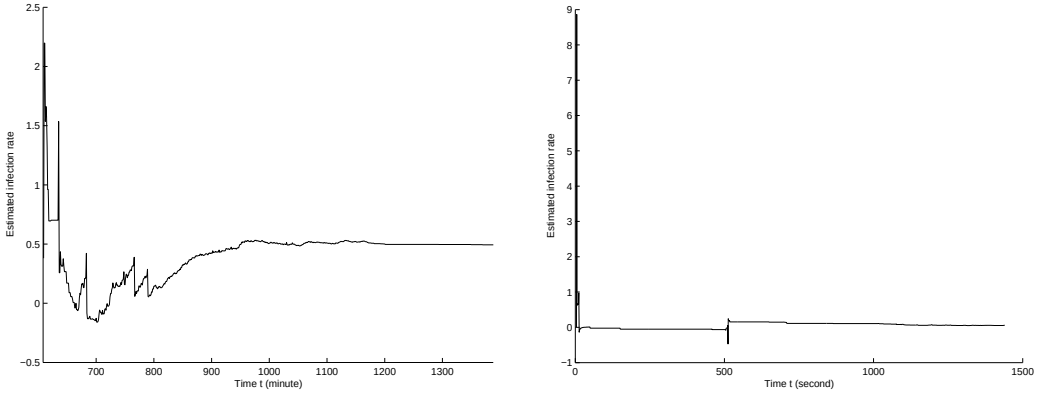
In [22], Zou et. al applied Kalman filters to the illegitimate scan traffic to the monitoring system to detect a worm at its early stage. In their approach, they used the idea of "detecting the trend, not the rate" of the illegitimate scans to the monitored networks [22]. In particular, they used Kalman filter to estimate the value of worm infection rate, denoted as $\alpha$. In practice, for each TCP and UDP port, an alarm threshold of the illegitimate scans is set. The Kalman filter is not activated until the scan traffic is over the alarm threshold for a certain number of times consecutively, denoted as $r$, e.g., three times. If the illegitimate scans are caused by non-worm noise, the estimated infection rate would oscillate around zero or oscillate around without a central point. Otherwise, if the scans are from a worm, the estimated value of infection rate output by Kalman filter would stabilize and oscillate around a *constant positive point* even if the estimated infection rate is not well converged [22]. They evaluated the performance of worm detection algorithms based on *simulation* and assumed that the propagation of worms follows an epidemic model [22].

In our experiments, we applied Kalman filter as designed in [22] to the data collected from our darknet. As our darknet uses unused IP addresses and only passively logs the incoming traffic without any interactions or outbound traffic, we consider all the ingress scans to our darknet as illegitimate. In our experiment, we tested Kalman filter-based detection algorithms to the real data of worms Code Red, Slammer and Blaster respectively. The threshold of illegitimate scans traffic was set based on the observations of our darknet during non-worm days.

In Code Red and Blaster experiments, we set the monitoring time unit as 1 minute as did in [22]. As [22] did, the Kalman filter was activated when the over-threshold events happen consecutively 3 times, i.e., $r = 3$.

Figure 1(a) shows the estimated infection rate $\alpha$ computed by Kalman filter in the worm Code Red. From the figure, we can see that $\alpha$ stabilizes and oscillates around a positive central point, which indicates an identification of worm. Figure 1(b) shows the output of Kalman filter that indicates the worm Blaster's activities. In both experiments, Kalman filter-based detection algorithm did not output any false worm alerts.

In the experiment of worm SQL Slammer, as [22] did, we set the monitoring time interval as 1 second because SQL Slammer worm propagates very fast. In the exper-




(a) Code Red: Kalman filter estimation of $\alpha$ (b) Blaster: Kalman filter estimation of $\alpha$

Figure 1: Kalman filter estimation of infection rate $\alpha$ in worm Code Red and Blaster

iment, we noticed that Kalman filter's output was a little ambiguous corresponding to scans to TCP port 443. We then varied the monitoring time interval as 1 second, 10 seconds and 30 seconds respectively to observe the Kalman filter's outputs.

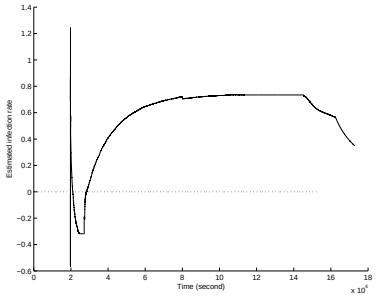
Figure 2 and 3 show the Kalman filter's estimation of infection rate of SQL Slammer worm and illegitimate non-worm scans to TCP port 443 respectively. In the Figure 3(a), we can see that the output of Kalman filter has a step-style stabilization that makes it a little ambiguous to tell if it is a worm activity or not. When we increased the monitoring time interval from 1 second to 10 seconds and 30 seconds, Figure 3(b) and 3(c) show that the corresponding Kalman filter's outputs stabilize at a *negative central point*, which indicates that scans to TCP port 443 are non-worm activities. In this case, although increasing the monitoring time interval can eliminate some ambiguity, at the same time, the detection time (i.e., the time that $\alpha$ begins to stabilize) is also deferred as shown in Figure 2. There exists a trade off between the elimination of ambiguous result and detection time. In this example, it also shows and confirms that the monitoring time interval is an important parameter in the Kalman filter-based worm detection system as mentioned in [22].

Considering the size of our monitoring network ( 25,600 unique inactive IP addresses), we may miss observing some activities that happen elsewhere. Therefore, our evaluation on Kalman filter-based detection algorithm is limited to relatively small networks instead of the larger monitoring system, e.g., networks with $2^{20}$ IP addresses, as discussed in [22].

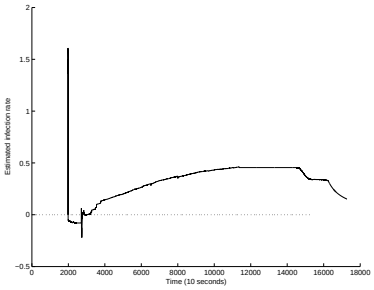
From our experience in these experiments, we notice that when deploying Kalman filter-based algorithms in our darknet or smaller local networks, we have to face the challenges of the sensitivity of Kalman filter's output to the monitoring time interval. When varying the monitoring time interval, the output of Kalman filter can be different. One possible solution is to run multiple Kalman filter-based detection engines with different time intervals parallel. However, it is also desirable to develop detection algorithms that can be practically deployed in local networks and less sensitive to monitoring time intervals.

### 3.3 Victim Number-based Approach

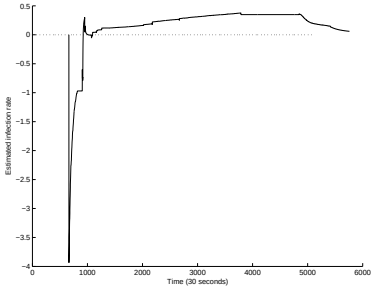
Wu et. al [21] proposed a worm early detection algorithm based on evaluating the increase rate of source addresses of illegitimate scans to the inactive IP space (e.g., the darknet in our study). A *victim* is defined as an IP address that sends a packet to an inactive IP space [21]. In practice, they use the "Two Scan Decision Rule", i.e., "two scans captured by the host leads to a victim" [11], to identify victims. In the detection, a detection system is deployed in an inactive IP space and tracks the increase number of victims. For a time tick $i$, if the rate of victim increase is over a threshold, an anomaly event is recorded by the detection system. If the anomaly event happens consecutively for more than a certain numbers, denoted as $r$, a worm alert is output. The threshold of victim increase rate at each time tick $i$ is computed as $\gamma * \sigma$ where $\sigma$ represents the standard deviation of victim increase during the prior $k$ unit time intervals, and $\gamma$ is a constant [21]. The parameters, i.e., $\gamma, k, r$, were




(a) SQL Slammer: Kalman filter estimation of $\alpha$, monitoring time unit: 1 second

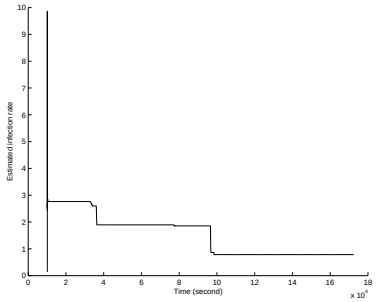



(b) SQL Slammer: Kalman filter estimation of $\alpha$, monitoring time unit: 10 seconds

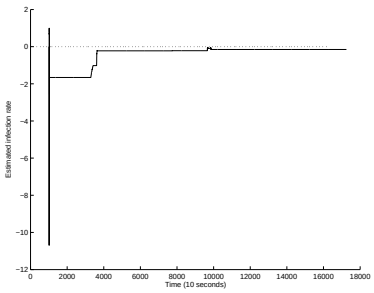



(c) SQL Slammer: Kalman filter estimation of $\alpha$, monitoring time unit: 30 seconds

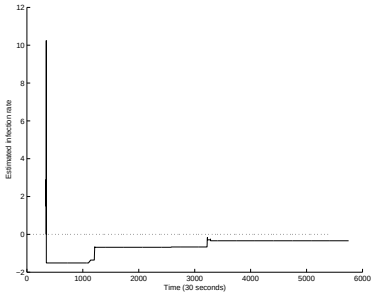
Figure 2: Kalman filter estimation of infection rate $\alpha$ in worm SQL Slammer with different monitoring time unit.




(a) Kalman filter estimation of $\alpha$ of TCP port 443, monitoring time unit: 1 second




(b) Kalman filter estimation of $\alpha$ of TCP port 443, monitoring time unit: 10 seconds




(c) Kalman filter estimation of $\alpha$ of TCP port 443, monitoring time unit: 30 seconds

Figure 3: Kalman filter estimation of infection rate $\alpha$ of TCP port 443 with different monitoring time unit.

based on normal traffic training and performance evaluation using background traffic traces from WAND research group [19].

In our experiments, we followed the detection algorithms as presented in [21]. Considering the inactiveness of our darknet compared with a normal network, we also accepted the traffic traces from WAND group as our background traffic, and used the same parameters (i.e., $\gamma = 3, k = 200, r = 10$) as used in [21] in our experiments.

In our experiments of Code Red, SQL Slammer and Blaster worms, victim number-based detection algorithm can correctly detect the worm propagation. The detection engine did not output alerts to non-worm activities in experiments of Code Red and SQL Slammer worms. However, in the experiment of Blaster worm, we noticed that there were multiple false positive alerts corresponding to scans to TCP port 139 and 445.

In Figure 4(a) and 4(b), the dotted lines represent the time points when the detection engine outputs worm alerts corresponding to scans to TCP port 139 and 445 respectively on August 6, 2003, 5 days before Blaster worm broke out on August 11, 2003. Based on our knowledge and analysis, we do not think the scans to TCP port 139 and 445 were worm-related on August 6, 2003. Therefore, we regard them as false positive alerts. In the figure, we can also see that some spikes of scan traffic did not trigger the detection engine to output alerts. The reason is that the detection algorithm is based on the anomaly detection of the *victim increase rate* instead of scan volume. It also requires the abnormal victim rate to increase a number of times consecutively (i.e., parameter $r$) before a worm alert is output.

As discussed in Section 3.2, we did not have enough data to evaluate the victim number-based detection algorithm on a large network (e.g., $2^{20}$-node networks) as suggested in [21]. We also realize that the limitation of our darknet size may suppress observation of some activities and therefore may degrade detection performance. Nonetheless, we desire a robust and stable worm detection algorithm that can be deployed on a relatively small network.

# 4 Destination-Source Correlation (DSC)

Most worms that we have observed so far have the following common characteristics. First, they generate a substantial volume of identical or similar traffic to the targets. However, the polymorphic worm may not follow this feature. Second, the worm infects vulnerable hosts for propagation. Third, many worms, e.g., Code Red and SQL Slammer, use random scanning to probe vulnerable hosts. Therefore, scans generated by this type of worm can reach inactive IP addresses. The current worm detection approaches, e.g. [22, 1, 21], mainly focus on the first and third characteristics of worms for detection. In our study, we believe the infection nature is the most important common feature of worms. Therefore, we designed worm detection algorithms based on investigation of worm infection.

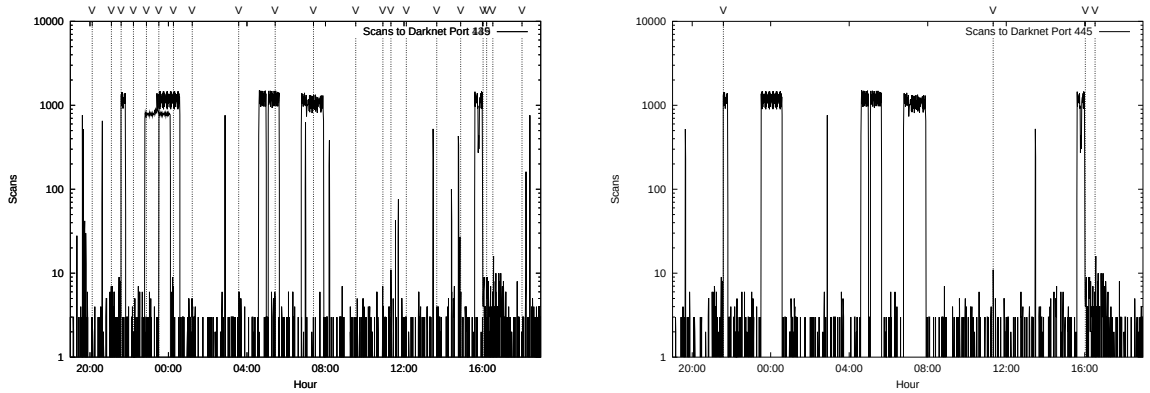
The worm infection feature can be summarized as follows. After a vulnerable host is infected by a worm on a port $i$ (i.e., the host is the destination of an early worm attack), the infected host will send out scans to other hosts targeting at the same port $i$ in a short time (i.e., the infected host is a source of new worm attack). For example, an infected host by Slammer worm on port 1434 also sends out scans to port 1434 of other hosts. Our detection model outputs worm alert when identifying this type of destination-source infection pattern.

In this section, we introduce our worm detection algorithm based on destination-source correlation of infection patterns.

## 4.1 DSC Algorithm

*General idea:* We keep a sliding window of previous network traffic (source and destination addresses of SYN and UDP traffic). Two general items are tracked: for each port witnessed in this traffic, we record the address of the inside destination host and the scanning source from the monitored network. The addresses can be IP, or MAC in order to defeat spoofing IP worms. If a source scan originates from a host that previously received a scan on an identical port, we increment a counter. If this counter passes a threshold trained for the network, we issue an alert.

We describe a simplified version using Bloom filters. For every port, we use three Bloom filters denoted as $D_{i-1}, D_i, S_i$, which track the destination addresses at time ticks $i-1$ and $i$ for scans directed at the network, and the source addresses for traffic (SYN and UDP) originating at time tick $i$ from the network. Thus, at every time tick $i$, we record the suspicious victim's IP address and the number of scans the victim sends out (i.e., scan rate). If the scan rate deviates from a normal profile, the




(a) Alerts corresponding to scans to TCP port 139

(b) Alerts corresponding to scans to TCP port 445

Figure 4: Alerts output by victim number-based detection algorithm on Aug. 6, 2003

suspicious victim is regarded as a real victim infected by a worm. We then output a worm alert and indicate the victim address and its scan rate. With technique of anomaly detection, we can achieve very low false positive rate. We discuss our anomaly detection technique in Section 4.3.2.

## 4.2 Analysis and Simulation

### 4.2.1 Different Scan Methods

Analytical Active Worm Propagation (AAWP[2]) model is a worm propagation model based on discrete time. Zou et. al [22] and Weaver [20] used epidemiological models to represent worm propagation. These two worm propagation models have similar performance [22, 2, 21]. We use the discrete time-based AAWP model for simulation and detection time analysis.

In the AAWP model, the number of infected hosts at time tick $i$ is shown in Eq.( 1), where $N$ is the total number of vulnerable machines in the Internet, $T$ is the size of IPv4 space used by the worm to scan, $s$ is the scan rate, $n_i$ is the number of infected hosts at time tick $i$. We have:

$$n_{i+1} = n_i + [N - n_i]\left(1 - (1 - \frac{1}{T})^{sn_i}\right) \qquad (1)$$

We analyze the performance of our DSC algorithm under three different worm scan strategies. We measure the detection performance in terms of infected percentage of whole vulnerable hosts when a worm alert is output.

*Random scan*: This is a normal scan strategy used by many worms, e.g. Code Red, Slammer. In the paper we assume the vulnerable hosts are uniformly distributed in the *whole scanning space* of worm as current worm propagation models do. For random scan, the number of vulnerable hosts in our monitored network is $V = \frac{ND}{T}$, where $D$ is the size of our monitored network, $T = 2^{32}$.

The chance of a certain node not being hit by a random scan is $1 - \frac{1}{T}$. We raise this chance to the power of $\sum_{j=0}^{i} sn_j$ to reflect all of the scans being performed up to time $i$. The chance of being hit at least once by time $i$ is therefore one minus this value. Therefore, the number of infected hosts in our monitored network at time tick $i$ is shown in Eq. ( 2), where $T = 2^{32}$.

$$v_i = \frac{ND}{T}\left(1 - \left(1 - \frac{1}{T}\right)^{\sum_{j=0}^{i} sn_j}\right) \qquad (2)$$

*Routable scan*: A routable scan worm only scans routable addresses which is around $T = 10^9$ [21, 24]. In the routable scan, the formulas to compute the infected hosts at time tick $i$ and the number of infected hosts in the monitored network are the same as in the *random scan* scenario. The only difference is that $T = 10^9$.

*Divide-Conquer scan*: This is an extension of routable scanning. In Divide-Conquer scan, when a worm from host $A$ infects a vulnerable host $B$, the worm at $A$ splits the scan target IP addresses into halves and transmits one

half of target IP addresses to victim $B$. Host $B$ then is responsible for scanning the half of IP space received from host $A$ [21, 24]. In this scan technique the worm does not revisit the addresses it already visited. So, the number of infected hosts at time tick $i$ is shown in Eq.( 3).

$$n_{i+1} = n_i + [N - n_i]\left(1 - (1 - \frac{1}{T - \sum_{j=0}^{i-1} sn_j})^{sn_i}\right) \quad (3)$$

The expected number of newly infected hosts in our monitored network at time tick $i$ is $\frac{ND}{T}$ times $(1 - (1 - \frac{1}{T-\sum_{j=0}^{i-1} sn_j})^{sn_i})$, the possibility of being hit at time $i$.

Since infected hosts will not be hit again [21], the total number of infected hosts in our monitored network at time $i$ is shown in Eq. ( 4), where $T = 10^9$.

$$v_i = \sum_{k=0}^{i} \frac{ND}{T}\left(1 - \left(1 - \frac{1}{(T - \sum_{j=0}^{k-1} sn_j)}\right)^{sn_k}\right) \quad (4)$$

#### 4.2.2 Simulation with Various Scan Methods

We simulate the worm propagation with the three scan techniques discussed above and evaluate the detection time of our DSC detection algorithm. We define the detection time as the time when at least one infected host is identified, i.e., the time when $v_i \geq 1$. (We assume that our DSC algorithm can detect a worm when the first victim is seen.)

Figure 5 shows the worm propagation using three different scan methods. It also indicates the detection time and corresponding infection percentage. We can see that our algorithm can detect worm at an early stage. Table 1 shows the detail information about the worm detection time and corresponding infection percentage of different scan methods using /12 and /16 networks respectively. Table 1 indicates that the detection performance of routable worm and Divide-Conquer worm are better than that of random scan worm. The reason is that the random scan worm uses the whole IP V4 space as scan target address, i.e., $T = 2^{32}$, while the other two types of scan worms only targets routable IP V4 space, i.e., $T = 10^9$. And in our assumption, all vulnerable hosts are uniformly distributed in the scanning space of worm As shown in computing the number of vulnerable hosts in a network, i.e., $V = \frac{ND}{T}$, the number of vulnerable hosts in a network in the random scan case is less than that of the routable or Divide-Conquer scan cases.

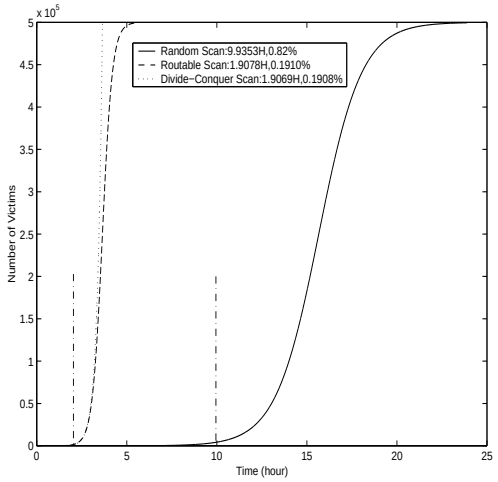
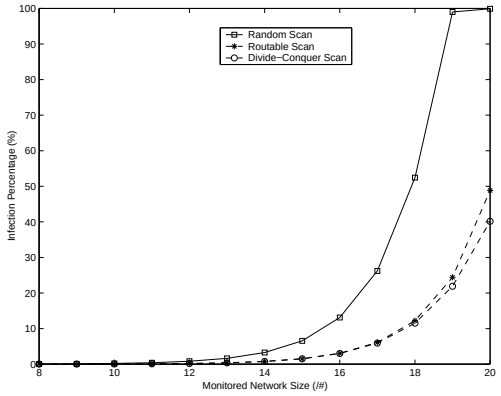



Figure 5: Detection time of worms spreading with three scan methods. Size of monitored network: a /12 network; hit list=1; N=500,000; scan rate=2/sec; death rate=0; patch rate=0

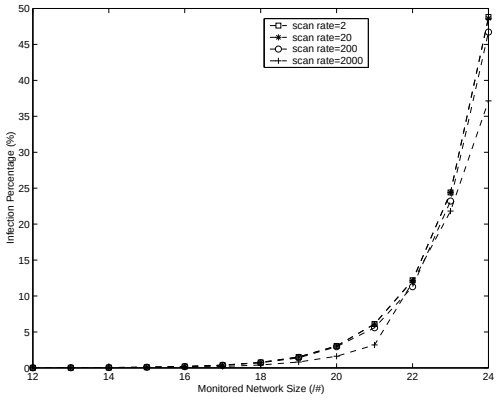
Figure 6(a) shows the corresponding infection percentage when we detect a worm using different scan techniques with the same scan rate as 2 scans/second under different sizes of monitored networks. From this figure, we can also see that our DSC algorithm is very effective in detecting routable and Divide-Conquer scan worms. When using a relatively small /16 monitored network, we can still detect these two kinds of worms before they infect 3.05% of whole vulnerable hosts in the Internet. For the random scan worm, using a /12 monitored network, we can detect a worm before 0.82% of whole vulnerable hosts in the Internet get infected.

Figure 6(b) shows the detection time using different scan rates and different sizes of monitored network for routable worm. Our detection time (i.e., the % of whole vulnerable hosts) with the DSC algorithm is nearly the same even when the scan rate is very high. This means our DSC algorithm is not sensitive to scan rate. Even for a very fast worm, the detection time is almost the same, in fact a little earlier.

There may exist a concern that it is uncertain to output a worm alert when only one victim is detected. We studied the relationship between the number of infected hosts in the monitored network and the detection time as shown in Figure 7. From Figure 7(a), we can see that in a /16 network, to have 2 routable worm victims detected, it requires an infection percentage of only 0.38%. It is still in an early stage. Figure 7(b) shows that given a certain

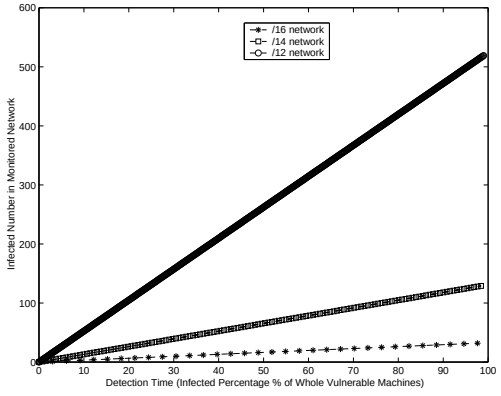



(a) Infection percentage when detecting a worm using different size monitored network and different scan methods: scan rate=2 scans/second

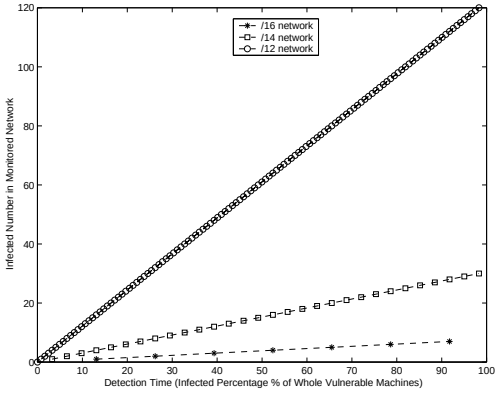



(b) Infection percentage when detecting a worm using different size monitored network and different scan rates (routable scans)

Figure 6: Effects on detection time by scan methods and scan rate, hit list=1, N=500,000, death=0, patch=0.




(a) Routable Worm




(b) Random Scan Worm

Figure 7: Relationship between detected number and detection time. hit list=1, N=500,000, scan rate=2 scans/sec, death=0, patch=0, monitored network size /16

Table 1: Worm detection time under different scan methods

| Scan Type | Scan Space | Monitor Space | Detection Time (sec) | Infection Percentage (whole Internet) |
|---|---|---|---|---|
| Random | $2^{32}$ | /12 | 35767 | 0.82% |
| Random | $2^{32}$ | /16 | 348243 | 13.11% |
| Routable | $10^9$ | /12 | 6868 | 0.19% |
| Routable | $10^9$ | /16 | 9670 | 3.05% |
| Divide-Conquer | $10^9$ | /12 | 6865 | 0.19% |
| Divide-Conquer | $10^9$ | /16 | 9624 | 3.01% |

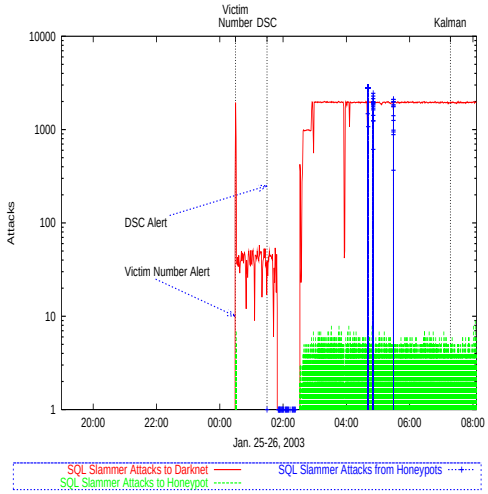



Figure 8: Detection time of three detection models to SQL Slammer worm

infection percentage, we will have more infected victims in a larger monitored network. For example, given 13.11% infection percentage of all vulnerable hosts in the Internet infected by a random scan worm, there are 16 victims detected in a /12 monitored network, 4 victims in a /14 network and 1 victim in a /16 network.

#### 4.2.3 Honeynet Verification

We applied our DSC algorithm to data collected from our honeynet during the breakout of SQL Slammer worm. Figure 8 shows that our algorithm can detect the Slammer worm at an early stage. Further, we can also determine all of the victims' IP information. The figure shows that the victim number-based detection algorithm outputs the worm alert earlier than the other two detection algorithms. However, considering the fact that the victim number-based algorithm has the weakness of producing false alerts as discussed in Section 3.3, and it cannot detect Divide-Conquer scan worms [21], we think that our algorithm performs better.

For CodeRed and Blaster worms, our DSC did not output any worm alerts because there was no infected worm victim in our honeynet. The reason is that our honeynet hosts were configured to be immune to these two worms.

### 4.3 Analysis and Discussion

#### 4.3.1 False Alarm Caused by Bloom Filter

Since we use Bloom filters to record IP addresses and inquire from them, we should analyze the false alarm caused by them. Bloom filters do not cause false negative. Therefore, we only need to evaluate the probability of false positive, the chance that some addresses are incorrectly counted as worm victims. The probability of false positive is in Eq.( 5), where $m$ is the size of bloom filter in bits, $n$ is the total number of hosts inside the monitored network, $k$ is the number of hash functions used. $I_i$ is the number of hosts inside the monitoring network that send out scans at time tick $i$, and $I_{i-1}$ represents the number of all inside hosts that are scanned at time tick $i-1$.

$$\begin{aligned}&((1-(1-\tfrac{1}{m})^{kI_{i-1}})(1-(1-\tfrac{1}{m})^{kI_i}))^k\\ &<((1-(1-\tfrac{1}{m})^{kn})(1-(1-\tfrac{1}{m})^{kn}))^k\\ &<(1-e^{-\frac{kn}{m}})^{2k}\end{aligned} \quad (5)$$

From this formula, we can see that if $\frac{m}{n}$ is big and we have a proper $k$, the number of hash functions used in Bloom filter, the probability of false positive would be very low. For example, when $\frac{m}{n}=30$, $k=12$, the probability false positive is less than $2.7225*10^{-12}$.

Assume we have a /16 network, it is usually impossible for every IP to be mapped to a real host. Assume we have

$n = 2^{14} = 16,384$ number of hosts in the monitored network and $\frac{m}{n} = 30$, then the size of Bloom filter is $m$=30 $n = 491,520$ bits $= 61,440$ bytes.

#### 4.3.2 False Alarm Caused by Threshold

In our detection algorithm, we need to detect the anomaly of outbound scanning rate from suspicious worm victims. We can apply heuristics of anomaly detection techniques to identify the anomaly. In practice, we first establish the normal profile of the outbound scan rate of hosts which have worm-like behavior (i.e., the host is first scanned, then sends out scans on the same port). In our study, we adapt an anomaly detection technique based on Chebyshev's inequality used in [9].

For a given random variable $x$, the Chebyshev inequality provides an upper bound on the probability that the value lies outside a certain distance from the variable's mean $\mu$. In the training phase, we approximate the mean $\mu$ and the variance $\sigma^2$ of the real scan rate distribution to each port by computing the sample mean $\mu$ and the sample variance $\sigma^2$ for the scan rate $s_1, s_2, ..., s_n$. In the detection phase, the anomaly detection engine assess the regularity of a value with scan rate $r$.

The intuition of applying Chebyshev inequality to anomaly detection is that if a value (denoted as $r$) deviates from the mean $\mu$ more than the normal deviation profile (i.e., $|x-\mu|$), this value $r$ is identified as anomaly. In particular, a Chebyshev inequality can be represented as Eq.( 6) where $t$ is a threshold, and $p$ is probability.

$$p(|x - \mu| > t) < \frac{\sigma^2}{t^2} \tag{6}$$

In our study, we substitute $t$ with the distance between the scan rate $r$ and the mean $\mu$ of the scan rate distribution (i.e., $|r - \mu|$). The resulting probability value of upper bound of $r$'s deviation from the mean is Eq.( 7).

$$p(|x - \mu| > |r - \mu|) < p(r) = \frac{\sigma^2}{(r - \mu)^2} \tag{7}$$

In our experiments, we used real traffic traces provided by the WAND group (traffic logs of /16 network) for training. The purpose of this training is to see whether we can establish a "normal" profile that includes some infection-like behaviors (i.e. a host is scanned first, then sends out scans to same port of other hosts in a short period of time), and what the false alarm rate may be if such profile is used for anomaly detection.

We selected a 65GB (compressed) trace sample, representing a continuous six and one half week trace between February and April 2001 at the University of Auckland uplink. We split the data into two parts for training and testing respectively, in particular, 80% of the trace was used as training data and the rest 20% is used for testing the false positive rate. We first established a normal profile of scan rate for every port immediately after infection-like behavior. Thus, if a host received traffic on port $i$, and then generated traffic to remote port $i$, we measured the outgoing rate. We focused on traffic from some representative TCP ports, i.e., 21, 22, 23, 25, 80, 139, 445 and UDP ports, i.e., 53, 1434.

Surprisingly, we did not find any *infection-like behaviors* on all selected ports except port 80. There were 25 infection-like behaviors total on port 80, however, the immediate outgoing scan rate after infection-like behavior to external port 80 proved to be very small, i.e., $\mu = 0.1, \sigma^2 = 0$. Testing results showed that the false positive rate is 0% in our WAND data experiment.

### 4.4 Advantages of Our DSC Algorithm

As soon as there is a host in our network infected by a worm, we can tell there is a worm. This is fast, especially for routable worms and Divide-Conquer worms. Current algorithms, e.g., Kalman and victim number-based, are effective against those using random scanning. For routable worms, Divide-Conquer scan worms or smart worms that only scan to real, in-use IPs, their algorithms may not be effective. In fact none of the current algorithms can detect *Divide-Conquer* scan worms. The Kalman filter study, for example, only used a /12 network to detect random scanning worms [22]. For routable worms, the study suggested upgrading to IPv6 [24]. Victim number-based approaches cannot detect Divide-Conquer scans worms [21]. But our DSC is very effective to these kind of scan methods.

The advantages of our DSC approach compared with others are the following. First, DSC algorithm aims to detect real worm victims with local network/honeynet, not just the scan traffic increasing problem. Second, our algorithm can identify the real worm victims inside a local network, which provides important information to local network administrators. Third, our algorithm is

effective to detect worm using non-random scan techniques, e.g., routable worm, Divide-Conquer or smarter worm that only scans to real IPs. Fourth, with distributed deployment of our detection algorithm, we can not only detect the worm in very early stage but can also determine the worm victims' distribution. Fifth, we can have accurate response to accurate individual victims. For example, we can block the port of real victim instead of the port of all hosts in a network. And more actively, we can immediately take actions (e.g. patching) to immunize the victims.

One major difference between our approach and other approaches, e.g., [22, 21], lies in the data sources. They proposed to apply the detection mechanisms to data collected from inactive IP addresses, e.g., darknet. Our approach can be used in production networks. Therefore, we believe our technique is complementary to other existing worm detection algorithms.

# 5 HoneyStat

We also propose a means of augmenting darknets using modified honeypots to enrich the information they yield.

## 5.1 Honeypots

Honeypots have been traditionally used to capture malicious code and provide security analysts with small, high value data sets [13, 10]. Honeypots focus on a particular security problem, like sampling a particular worm or virus, or monitoring live interactive sessions with hackers to learn their techniques and capture their tools. They are traditionally labor intensive, sometimes requiring hours to analyze only minutes of captured activity. While researchers are starting to find wider roles for honeypots, their use still requires intensive-hand verification and analysis. We propose the use of honeypots to gather statistical information about network activity and detect worm propagation.

## 5.2 Possible Honeypot Uses

We begin with a few intuitions about honeypot analysis, and how they relate to worm detection.

First, let us consider the simple scenario where a honeypot "victim" receives network traffic from a remote host, and then begins to send out traffic to different machines. One might suppose that the traffic sent to the honeypot was a worm. Or, it could the actions of a "live" hacker. Our suspicions become stronger if there are similarities in the traffic sent to and generated by the honeypot. For example, if the destination ports are always the same, or if the payload is substantially similar, one can suspect malware.

Further proof comes when one observation follows another–when multiple honeypots are compromised in a pattern. This is not unlike how security researchers currently spot fast breaking worms and viruses. Using e-mail, IRC or instant messaging, researchers compare local observations, and quickly spot emerging patterns. We propose the use of specially modified honeypot sensors to automate this process, and help detect worm outbreaks.

## 5.3 Model of Infection

A key assumption in our monitoring system is that the worm infection can be described in a systematic way. We first note that worms often generate network activity during the initial infection phase. The Blaster worm is instructive.

Blaster consists of a series of modules designed to infect a host. The first module was based on a widely available RPC DCOM exploit that spawned a system shell on a victim host. The "egg" payload of the worm was a separate program (usually "msblast.exe") that underwent many revisions and changes. So, while Blaster broke out on August 11, 2003, there were many other attack tools based off the original DCOM exploit. Blaster was just the first to automate replication.

The infection process, illustrated in Figure 9, begins with a probe for a victim providing port 135 RPC services. The service is overflowed, and the victim spawns a shell listening on a port, usually 4444. (Later generations of the worm use different or even random ports.) The shell remains open for only one connection and closes after the infection is completed. The shell is used to instruct the victim to download (often via tftp) an "egg" program. The egg can be obtained from the attacker, or a third party. The time delay between the initial exploit and the download of the egg is usually small. Exploits that wait a long period to download the egg risk having the service restarted, canceled, or infected by competing worms (e.g., Nachi). Still, some delay may occur between the overflow and the "egg" transfer. All

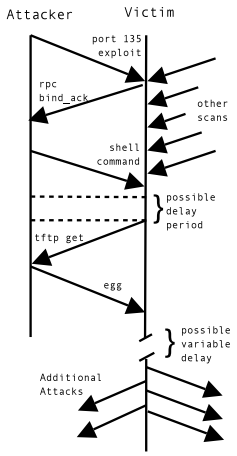



Figure 9: General pattern of Blaster worm attack. Because of modular worm architectures, victims are first overflowed with a simple RPC exploit, and made to obtain a separate worm "egg", which contains the full worm. The network activity between the initial overflow and download of the "egg" constitutes a single observation. Multiple observations allow one to filter out other scans arriving at the same time.

during this time, other network traffic may arrive.

Traditional worm detection models deal with worm infection at either the start or end of the cycle shown in Figure 9. For example, models based on darknets consider only the rate and origin of incoming scans–the traffic at the top of the diagram. The DSC model also considers scans, but also tracks outgoing probes from the victim–traffic from the bottom of the diagram. The activity in the middle–from initial infection to subsequent attack–has a distinct enough form that one can track it in a network.

Even if no buffer overflow is involved, as in the case of mail-based worms and LANMAN weak password guessing worms (e.g., pubstro worms), the infection still follows a general pattern: a small set of attack packets obtain initial results, and further network traffic follows, either from the egg deployment, or from subsequent scans. Thus, our proposal is not limited to Blaster, but uses it as an illustrative example.

### 5.4 HoneyStat Monitoring

We propose a new variation of honeypots, called HoneyStat, to provide early detection and local response for worm outbreaks. Unlike traditional honeypots, HoneyStat nodes are not necessarily used for tracking live hackers. (Of course, they could provide this secondary use). Instead, HoneyStat nodes are used to gather statistical data information about network attacks.

The key idea is to deploy large numbers of honeypots on virtual machines, each multihomed to cover a large address space. When a honeypot is attacked, the intrusion is detected, and the virtual machine image is reset. Statistical properties of the attack are recorded, such as which ports were active just before the attack was discovered. These observations are then combined with other observed attacks. A statistical analysis attempts to discover patterns indicative of a worm outbreak.

So, while honeypots trap hackers, HoneyStat nodes track worms. This approach differs from DSC in many ways. First, while DSC uses real network or honeypot traffic, HoneyStat works only on special honeypot networks. Second, while DSC matches the same port being used for source and destination scans, HoneyStat matches arbitrary pairs of ports. Potentially DSC will notice victims earlier than HoneyStat, provided the infection vector uses the same ports. However HoneyStat can potentially provide more information about the worm behavior.

In more detail, we describe the deployment of HoneyStat nodes:

First, an emulator (e.g., Bochs or VMWare) is used to expose a virtual machine to the outside network. The guest operating system in the emulator is configured to provide a wide range of services. Vendor patches for any of the services are not applied, except in rare cases. An emulator is used instead of a live machine to facility rapid resets of the guest OS image. This lets one deploy as many honeypots as possible, the goal being to interact with suspicious traffic as much as possible.

Additionally, the honeypots are configured to direct as much traffic as possible back towards other honeypots and darknets. For example, some worms such as the recent MyDoom worm spread via e-mail, and would be missed by simple darknet scan observation techniques. To track worms traveling "under the radar", the honeypots are configured with honeytokens: e-mail address books for bogus users at other honeypots and darknet nodes.

Second, the honeypot is configured to not generate any outward network traffic, e.g., windows NBT probes. The honeypot should remain silent, since originating out-

ward traffic constitutes a trigger event. In our model, a honeypot is deemed active or infected when it sends out SYN or UDP traffic.

Third, a separate process monitors the virtual machine's disk image, and network activity. If outgoing connections are made from the honeypot, an "HoneyStat event" is recorded, and after a suitable short period of time (e.g., just long enough for the "egg" to arrive, or after writes complete to key directories and registry settings), the honeypot is reset, using an archived copy of the original disk image, or a round-robin style set of identical images.

Fourth, once a HoneyStat event occurs, a range of network traffic just before the event is saved, e.g., 5 minutes before the honeypot "wakes up". The traffic is combined with other observations, and if a logistical analysis of the data identifies a common cause, an alert is issued. The intuition is this: some recent event may explain why this honeypot woke up. We might not find the explanation in all events, so a statistical analysis is used to sift out the "liveware" from the wormware.

Finally, once the honeypot event is recorded, the disk image is reset to a fresh copy of the Guest OS. (In our prototype, we used a round-robin set of identical images, so that service was restored in seconds, and then used a slower thread to restore infected images.) Additionally, one might make copies of the files alerted or created by the infection.

Thus, HoneyStat nodes are used to scrutinize the random scanning and attacking conducted by worms. If one is fortunate, or the target of a directed worm, a honeypot could potentially provide very early detection, perhaps even before other rate-based algorithms stabilize.

We have deployed a working prototype of such a system, using commodity hardware for a single IP. We are able to measure the mean time before an infection occurs, and also record the type of infection that took place. Without supervision, the system gathered a complete collection of Blaster worm variants, scanning kits, pubstro worms, and a few root kits. We are currently expanding this monitoring to dozens of IPs.

More importantly, we were able to automate a detection system that recognized patterns of worm activity. This effectively turns honeypots into a stream of very sensitive monitoring nodes.

### 5.5 Statistical Analysis

Using the data collected from the 5,000-node network, we can locate events where enough network data was recorded to infer how HoneyStat performs. We therefore provide a proof-of-concept of the algorithm, and demonstrate its general effectiveness and resistance to false positives. Further study can validate this preliminary observation.

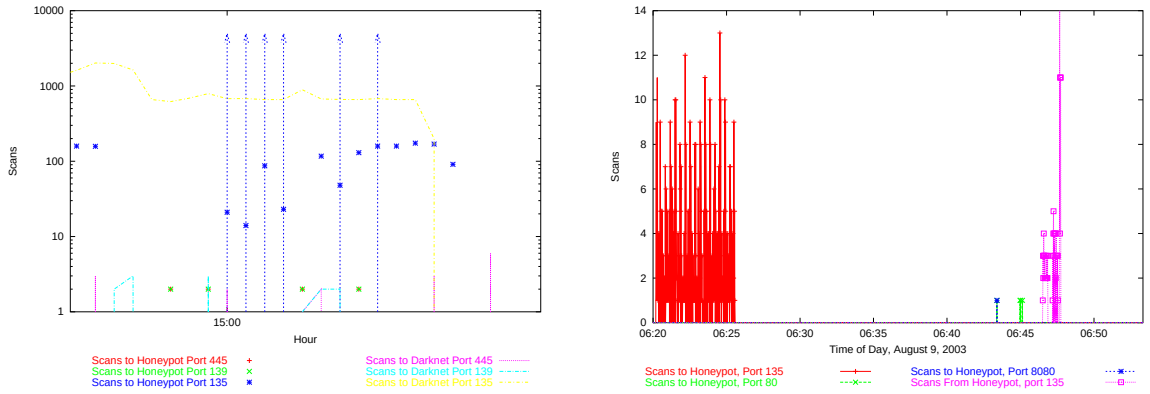
Figure 10(a) shows several HonetStat nodes that have been compromised. A search of the previous minutes of traffic yields a list of suspect ports. Under our stated model for worm infection, traffic to one of these ports might explain the honeypot's active state.

We chose not to use packet counts in our analysis, because this varied greatly in the trace data. Since one infective packet is enough to compromise a host, we instead chose to model the *dichotomous* change in the honeypot state. The node was either *dormant* or *active*, while the prior network provided continuous data with their time differential.

Our objective is to detect *zero-day* worms that have not known signatures. Without the ability to perform pattern matching, we apply statistical technique to analyze and correlate events in order to detect worm activities. In particular, we apply logistic analysis [3] to conduct port correlation. Logistic regression is a non-linear regression model that analyzes the relationship between a *dichotomous* variable (i.e., *dependent* variable) and a set of *independent* variables (i.e.,*explanatory* variables). In our study, we define the state of honeypot as the dichotomous variable and other prior port activities as independent variables. We then apply logistic analysis to select factors (or port activities) that can explain honeypot activations. The basic form of the logistic model expresses binary expectation of honeypot status, $E(Y)$, as seen in Eq. (8).

$$E(Y) = \frac{1}{1+e^{-Z}}, Z = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n + \epsilon \quad (8)$$

Here, $Y$ represents the boolean HoneyStat node state, i.e., *active* or *dormant*, and $\beta_i$ is the regression coefficient corresponding to the $X_i$ variable, or individual port. Only those ports that had scans just before the honeypot event are considered. The value used for each variable $X_i$ is the inverse of the time difference between the observation and the honeypot event. Thus, ports that




(a) HoneyStat Worm Detection

(b) HoneyStat Observance of Non-Worm Events

Figure 10: a) HoneyStat worm detection for Blaster. Upward arrows, not to scale, indicate the presence of outgoing traffic from the HoneyStat nodes. We take traffic from five minutes prior to these points and analyze what port activity could explain the honeypot activity. A logit analysis shows that prior scans to port 135 explains these episodes–effectively identifying the blaster worm. b) Avoiding false positives. Here, we see a Trojaned honeypot node becoming active. However, since this event is seen only in isolation (one honyepot), it does not trigger a worm alert.

had traffic just before the honeypot woke up are favored. (This is consistent with our model of worm infection.)

As shown in Table 2, a backward likelihood ratio analysis of the honeypot data shows that of all the variables, port 135 explains the tendency of honeypots to become active. (In our particular example, one can even visually confirm in Figure 10(a) that honeypot activity took place right after port 135 traffic arrived.) The Wald statistic indicates whether the $\beta$ (i.e., the coefficients shown in Eq. 8) statistic significantly differs from zero.

The significance column in Table 2 is the most critical for us to filter out unrelated port activities to the active state of honeypot. In our experiment, we set up the threshold of significance level as $5\%$. The *lower* the significance score of a port, the *more* chance that the port activity has influenced the activation of honeypot. Therefore, any port activity with a significance score less than the threshold (i.e., $5\%$) is regarded as a significant explanation to the honeypot state. As shown in Table 2, we can filter out port 80, 8080 and 3128 from explanatory variable list. In Wald statistic, a large coefficient $\beta$ causes a larger standard error. Therefore, we also filter out port that has a larger standard error. In our case, as shown in Table 2, we can rule out port 139 and port 445 as likely "causal" ports. In Table 2, port 135 has the most explanatory power to the active state of hoeypot, i.e., worm activity.

In this case, the logit analysis performs two useful tasks. First, as shown in Table 2, the high standard error value of port 139 rule it out as a likely candidate port. This outcome of the HoneyStat analysis contrasts with other algorithms, e.g., Victim Count[21] that generated false alarms on port 139 scans. HoneyStat was able to use the inconsistent time differences between scans to port 139 to rule it out as a causal factor. Similarly, the only other likely port identified by other algorithms, port 445, is likewise eliminated.

Second, the logit analysis also suggests a possible cause for the observed honeypot activities. In this case, the higher statistical analysis of port 135 suggests that unit changes in port 135 traffic greatly improves the odds that a honeypot becomes active. While we might want more samples and certainty, we can at the very least rank likely ports in an alert. Thus, HoneyStat reports *a theory* of worm activation, and not merely the *presence* of a worm. Other information, such as rate of scans, can be obtained from the traffic logs captured for the logit analysis.

The key to the success of this analysis is of course a suitable population size. If only a few HoneyStat events are recorded, there may not be enough data to identify a candidate. In the example above, seven honeypot events were observed–six due to blaster worms, and one due to and older infection. The "noise" contributed by the seventh event was not enough to diminish the confidence interval of the reported statistics. One could easily imagine

Table 2: Logit Analysis of Multiple HoneyStat Events

| Variable | $\beta$ | Standard Error | Wald | Significance |
|---|---|---|---|---|
| port_80 | -17463.185 | 2696276.445 | .000 | .995 |
| port_135 | 3.114 | .967 | 10.377 | .001 |
| port_139 | 1869.151 | 303.517 | 37.925 | .000 |
| port_445 | -1495.040 | 281.165 | 28.274 | .000 |
| port_3128 | -18727.568 | 9859594.820 | .000 | .998 |
| port_8080 | 10907.922 | 10907.922 | 6919861.448 | .999 |
| constant | .068 | 1.568 | .210 | 1.089 |

situations where more noise does occur. So, if HoneyStat is to depend on more than good fortune, there should be many independent observed events.

So, the HoneyStat analysis provides the following insights: First, how many honeypots were compromised, and in what time frame. Second, whether factors appearing in many of the events can be eliminated. (E.g., port 139 and 445 look "hot", but there are too many observations where these ports are silent.) Third, what factors (ranked) can explain odds changes in the honeypot state. Instead of merely issuing an alert (e.g., "you've got worms") the HoneyStat analysis suggests a possible infection vector (e.g., "there's a worm, and with X confidence, the worm enters on port Y and targets on port Z"). In our example, the HoneyStat analysis reports with some confidence that a worm is using port 135 to infect a number of machines.

Statistical analysis, of course, can be erroneous, particularly for small sample values. Without an exhaustive data set to test HoneyStat, we can speculate about failure conditions. First, when attackers flood the network with tremendous amounts of extraneous traffic to confound analysis. In such a scenario, however, attackers will have to sacrifice bandwidth needed for worm propagation. Second, when worms take a tremendous amount of time to download an "egg", thereby delaying the trigger event for the worm. Recall that we noted a short period of time between the buffer overflow and the transfer of the complete worm. If attackers stretch this time period out, it would certainly add more noise to the sample space. However, it would also significantly slow the progress of the worm itself, since infection could not complete until the payload is obtained. Hanging worms also face attritional factors such as reboots, restarts, and the management of hung programs that do not seem to return from function calls. Future work may model this factor, but we see it as a challenge for worm writers to overcome.

We located a few situations in the logs where HoneyStat could fail in the absence of other corrective observations. Figure 10(b) shows a particular time frame where a honeypot started generating port 135 activity. Previous activity included incoming port 80 and 8080 scans, and some very distant activity on port 135 outside of the sample range. As a single observation could conclude that port 80 explains the honeypots activities. But with just one observation, there's not enough data to draw such a conclusion. As it turns out, this particular honeypot was infected days earlier, and used on-and-off for IRC relay, scanning, and testing malware. When this observation is added to the other honeypot events, it does not significantly influence the outcome.

Admittedly, the data set we analyzed had a few fortunate occurrences–the honeypots all reacted to the worm, and minimal noise was discarded from the infection model by the logit analysis. To *reliably* reproduce this for future worms, one must deploy a sizeable number of minimal honeypots. Using strategies such as virtual machines and large multihoming, one can efficiently span a suitable address space. The size of the data collection required to provide early detection is no different than the needs of DSC (see Section 4) or any other algorithm.

## 6 Conclusion and Future Work

In this study, we reviewed some of the monitoring strategies used for large networks. The need for a global monitoring system is clear. Likewise, the need for local detection and response is also obvious. However, many of the strategies are difficult to apply to local networks. In particular, the Kalman filter and victim number-based approaches can be difficult to manage on smaller networks.

We propose two algorithms tailored for local network

monitoring needs. First, the DSC algorithm focuses on the infection relation, and tracks real infected hosts (and not merely scans) to provide an accurate response. Second, the HoneyStat system provides a way to track the short-term infection behavior used by worms. Potentially, this may provide a basis for statistical inference about a worm's behavior on a network.

We examined all of these algorithms in light of a large data set. Nonetheless, we note for future work, the possibility of a) modifying DSC to use an adaptive time window, adjustable with current scan rates; b) use more real network traces to train and verify DSC; c) gather more statistical data about honeypot interactions with early worm outbreaks; and d) consider strategies to improve the data resources available to a CDC-style center, in light of these improved local monitoring capabilities.

## 7 Acknowledgments

We gratefully thank Mr. Cliff C. Zou and Mr. Jiang Wu at University of Massachusetts, Amherst for providing us their source codes, suggestions and helpful discussions on the research work. We also thank Mr. John Levine and Prof. Henry Owen at the Georgia Institute of Technology for substantial honeypot logs.

## References

[1] V.H. Berk, R.S. Gray, and G. Bakos. Using sensor networks and data fusion for early detection of active worms. In *Proceedings of the SPIE AeroSense*, 2003.

[2] Z. Chen, L. Gao, and K. Kwiat. Modeling the spread of active worms. In *Proceedings of the IEEE INFOCOM 2003*, March 2003.

[3] D.W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. Wiley-Interscience, 2000.

[4] SANS Institute. http://www.sans.org.

[5] R.E. Kalman. A new approach to linear filtering and prediction problems. *Transaction of the ASME–Journal of Basic Engineering*, March, 1960.

[6] J.O. Kephart, D.M. Chess, and S.R. White. Computers and epidemiology. 1993.

[7] J.O. Kephart and S.R. White. Directed-graph epidemiological models of computer viruses. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 343–359, 1991.

[8] J.O. Kephart and S.R. White. Measuring and modeling computer virus prevalence. In *Proceedings of IEEE Symposium on Security and Privacy*, 1993.

[9] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS'03)*, October 2003.

[10] John Levine, Richard LaBella, Henry Owen, Didier Contis, and Brian Culver. The use of honeynets to detect exploited systems across large enterprise networks. In *Proceedings of the 2003 IEEE Workshop on Information Assurance*, 2003.

[11] D. Moore. Code-red: A case study on the spread and victims of an internet worm. http://www.icir.org/vern/imw-2002/imw2002-papers/209.ps.gz, 2002.

[12] D. Moore. Network telescopes: Observing small or distant security events. http://www.caida.org/outreach/presentations/2002/usenix_sec/, 2002.

[13] Honeynet Project. Know your enemy: Honeynets. http://project.honeynet.org/papers/honeynet/.

[14] Niels Provos. A virtual honeypot framework. http://www.citi.umich.edu/techreports/reports/citi-tr-03-1.pdf, 2003.

[15] E. Skoudis. *Counter Hack*. Upper Saddle River, NJ: Prentice Hall PTR, 2002.

[16] Lance Spitzner. *Honeypots: Tracking Hackers*. Addison Wesley, 2003.

[17] S. Staniford. Code red analysis pages: July infestation analysis. http://www.silicondefense.com/cr/july.html, 2001.

[18] S. Staniford, V. Paxson, and N.Weaver. How to Own the Internet in Your Spare Time. In *Proceedings of 2002 Usenix Security Symposium*, 2002.

[19] Waikato internet traffic storage. http://wad.cs.waikato.ac.nz/wand/wits/index.html.

[20] N. Weaver. Warhol worms: The potential for very fast internet plagues. http://www.cs.berkeley.edu/ nweaver/warhol.html.

[21] J. Wu, S. Vangala, L. Gao, and K. Kwiat. An efficient architecture and algorithm for detecting worms with various scan techniques. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS'04)*, February 2004. to appear.

[22] C. C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for internet worms. In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS'03)*, October 2003.

[23] C. C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis. In *Proceedings of 9th ACM Conference on Computer and Communications Security (CCS'02)*, October 2002.

[24] C.C. Zou, D. Towsley, W. Gong, and S. Cai. Routing worm: A fast, selective attack worm based on ip address information. Technical Report TR-03-CSE-06, Umass ECE Dept., November 2003.